Maps API for JavaScript

Migration Guide

Version 3.0.12.0

Contents

Legal Notices

Trademark Acknowledgements

HERE and Nokia are trademarks or registered trademarks of Nokia Corporation.

Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

This content is provided "as-is" and without warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, satisfactory quality and non-infringement. HERE does not warrant that the content is error free and HERE does not warrant or make any representations regarding the quality, correctness, accuracy, or reliability of the content. You should therefore verify any information contained in the content before acting on it.

To the furthest extent permitted by law, under no circumstances, including without limitation HERE's negligence, shall HERE be liable for any damages, including, without limitation, direct, special, indirect, punitive, consequential, exemplary and/ or incidental damages that result from the use or application of this content, even if HERE or an authorized representative has been advised of the possibility of such damages.

Document Information

Product	
Name:	Maps API for JavaScript
Version:	Version 3.0.12.0

Document	
Name:	Maps API for JavaScript Migration Guide
Id:	5b00bd2-1434633034
Status:	FINAL
Date:	2015-Jun-18, 13:19 (GMT)

Chapter 1

Overview

Topics:

This guide provides information on difference between version 2.x of the HERE Maps API for JavaScript and version 3.x. The aim of the guide is to facilitate migration of existing program code to the latest version of the API.

The guide is intended exclusively for developers, who have previously used version 2.x of the API.

What Has Changed

Version 3 of the HERE Maps API for JavaScript represents a major redesign and can be viewed as a completely new API. While many of the core concepts will be familiar to users of the Maps API 2.x, the changes affect almost all aspects of the API, including signatures and class structure.

Mobile Devices

The new JavaScript API brings a number of modifications that we were not able to implement in the context of the JavaScript API 2.x.

The original Maps API for JavaScript targeted mainly desktop environments, catering for a wide range of use cases from simple maps to fleet tracking applications. The redesign is a response to the enormous increase in mobile device usage and the customer demand to run map applications on those devices.

HTML5

Another factor is the HTML5 technology, which is now widely supported by most browsers and environments: it has made large parts of the old API obsolete. HTML5 and related technologies (SVG, CSS3, EcmaScript 5, etc.) allow us to reduce the code size of the API by removing normalization layers and shims.

One API

Version 3.x brings together the Enterprise and Consumer versions of the JavaScript API in one API for everyone, with all the features available to both the consumer-facing customers and enterprise customers.

Benefits of Maps API 3.x

The new API:

- enables map application development across a wide range of browser environments – both mobile and desktop
- offers one consolidated library comprising all features of the Enterprise Edition and the Standard Edition
- takes full advantage of the latest browser technologies, enabling applications to tackle use cases of arbitrary complexity

- is optimized for package size, faster load times and better execution performance
- is modular, allowing developers to load only the code the application requires
- offers full access to the HERE REST APIs including the Map Tile API, Geocoder API, Routing API, Enterprise Routing API and Places API
- integrates StreetLevel Imagery for interactive 360° street panoramas

Chapter 2

Guide

Topics:

The following articles summarize the differences between the two versions of the Maps API for JavaScript and illustrate the migration process through examples.

Summary of Differences

The table below shows a summary of the most important differences between the Maps API for JavaScript 3.x and Maps API for JavaScript 2.x, focusing on key concepts. Furthermore, in the Maps API for JavaScript 2.x, functional features were made available mainly though components and map objects, whereas in Maps API for JavaScript 3.x, functional features are distributed across the `core` module and `extensions` which do not need to be loaded unless your application requires them.

Table 1: Differences between 2.x and 3.x

Concept	Here Maps API 2.x	Here Maps API 3.x
Loading of API features	Loaded by automatic feature loader	Loaded as extension modules
Access to HERE Platform Services	Implicit access through a manager	Explicit access through a service
Map instance	Created as a display (object)	Created as a map (object)
Markers, icons and graphics	Implemented as DOM elements	Implemented as DOM and Bitmap markers
Extended functionality	Accessible through components	Accessible through extension libraries
Events	Supported as part of the map class	Supported by the `mapevents` extension
UI controls	Delivered as map components	Delivered through the `UI` extension

Browser Support

Due to the fact that version 3.x of the API takes advantage of HTML5 technologies to provide better performance, mobile device support and usability, it works only in browsers that support HTML5. The Maps API for JavaScript 2.x did not require HTML5 and operated in older browsers, including Internet Explorer 7 and 8.

Table 2: Browsers supported by the HERE Maps API for JavaScript

Browser	Maps API 2.x	Maps API 3.x
Internet Explorer (desktop)	IE7+	IE9+, optimized for IE10+
Internet Explorer (Windows Phone)	IE9+	IE9+, optimized for IE10+
Firefox (desktop)	3.6+	optimized for latest
Chrome (desktop)	12+	optimized for latest

Browser	Maps API 2.x	Maps API 3.x
Chrome (Android)	Android 2.2+*	Android 2.2+, optimized for Android 4.4
Safari (MacOSX)	5+	6+, optimized for 7
Safari (iOS)	iOS4+	iOS6+
Android default browser / WebView	no support	Android 2.2+
iOS WebView	no support	iOS 6+

Loading API Features

Version 2.x of the JavaScript API uses an automatic feature loading mechanism which – depending on a number of parameters – downloads JavaScript files appropriate to the feature description and the user's browser environment.

The new API does not include a feature loader, instead it is available as a set of modules that contain feature implementations and can be loaded individually. For an overview of the available API modules and their dependencies, please refer to the developer's guide.

This new approach offers a number of advantages compared to the old version of the API:

- Developers have complete control over the features they would like to include in the application.
- Developers are free to choose the mechanism that loads the API files (synchronously as `script` tags, asynchronously, loading via require.js, etc).
- There is a clear mapping between the module files and the namespaces (for example, the module `mapsjs-mapevents.js` adds the `mapevents` namespace).

Loading API 2.x

In version 2.x, it was enough to include the single `API loader` script `jsl.js` and specify the required modules by modifying the URL:

```
<script src="http://js.api.here.com/se/2.5.4/jsl.js?with=maps,routing" type="text/
javascript" charset="utf-8"></script>
```

This script downloads a set of JavaScript files necessary to access the 'map' and 'routing' features, eight files in all.

Loading API 3.x

Version 3.x, replaces the loader concept with explicit loading of the required API modules. The core map rendering functionality is delivered by the `core` module (`mapsjs-core.js`). All other modules

extend the functionality of the `core` and you can choose which of them to load to implement your application.

The following example loads the `core` module and then the `service` module (`mapsjs-service.js`) to enable access to the HERE Platform services such as the HERE Map Tile API or the HERE Routing API.

Note that because the `service` module depends on the `core` module, it must be loaded after `core`.

```
<script src="http://js.api.here.com/v3/3.0/mapsjs-core.js" type="text/javascript"
 charset="utf-8"></script>
<script src="http://js.api.here.com/v3/3.0/mapsjs-service.js" type="text/javascript"
 charset="utf-8"></script>
```

The above example illustrates the key principle that only the modules that you explicitly request are downloaded and can be accessed by the application.

Accessing HERE Services

In version 3.x of the Maps API for JavaScript, the `service` module (`mapsjs-service.js`) consolidates interfaces to all the HERE back-end services (REST APIs), offering the `Platform` class as the common access point to services.

The `Platform` class holds the global settings for all the services the application may need and acts as a factory for service stubs, mapping them directly to the appropriate HERE REST APIs. This makes working with the services easy: the application deals only with the `Platform` object. By contrast, version 2.x provided the `Settings` class with all the global configuration settings for the API and a number of components and classes through which to access the services.

Accessing Services in 2.x

The entire API version 2.x was closely coupled with the HERE Service Platform, which resulted in the distribution of service-related functionality in different parts of the API. Below is an example that sets up access to Routing and the Geocoder in version 2.x – in both cases, the code uses a Manager class in different namespaces:

```
// Set global settings for platform access
nokia.Settings.set("app_id", "<!-- your app id -->");
nokia.Settings.set("app_code", "<!-- your token -->");+
nokia.Settings.set("serviceMode", "cit");
nokia.Settings.set("secureConnection", "force");

var map = new nokia.maps.map.Display(document.getElementById('map-container'), {
  // Implicitly access the HERE Map Tile API:
  baseMapType: nokia.maps.map.Display.SATELLITE
});

// Implicitly access the HERE Routing API or the HERE Enterprise Routing API when
```

```
// using the Enterprise Edition of the API:
var router = new nokia.maps.routing.Manager();

// Implicitly access the HERE Geocoder API:
var geocoder = new nokia.maps.search.Manager();
```

Accessing Services in 3.x

In the new API, the code is clearly decoupled from the HERE Services. This introduces more clarity, while at the same time providing a common and convenient way to communicate with the HERE Service Platform and to use the service features easily and flexibly. The following example sets up access to the Map Tile API, Routing and the Geocoder through an instance of the `Platform` class:

```
var platform = new H.service.Platform({
  app_id: '<!-- your app id -->',
  app_code: '<!-- your token -->',
  useCIT: true,
  useHTTPS: true
});

// Explicitly access the HERE Map Tile API:
var maptiler = platform.getMapTileService();

// Explicitly access the HERE Routing API:
var router = platform.getRoutingService();

// Explicitly access the HERE Enterprise Routing API.
var enterpriseRouter = platform.getEnterpriseRoutingService();

// Explicitly access the HERE Geocoder API:
var geocoder = platform.getGeocodingService();
```

Initializing the Map

Map initialization in version 3.x of the API is very similar to map initialization in version 2.x. However, there are some small but fundamental differences between the version 2.x class `nokia.maps.map.Display` and the class `H.Map` in version 3.x.

Map Initialization in 2.x

The following script shows map initialization in version 2.x of the API:

```
nokia.Settings.set("app_id", "<!-- your app id -->");
nokia.Settings.set("app_code", "<!-- your token -->");

var map = new nokia.maps.map.Display(document.getElementById('map-container'), {
  zoomLevel: 4,
  center: new nokia.maps.geo.Coordinate(-25.363882,131.044922)
});
```

Map Initialization in 3.x

The script below shows how to initialize a map in version 3.x:

```
var platform = new H.service.Platform({
  app_id: '<!-- your app id -->',
  app_code: '<!-- your token -->'
});
var defaultLayers = platform.createDefaultLayers();
var map = new H.Map(document.getElementById('map-container'), defaultLayers.normal.map, {
  zoom: 4,
  center: new H.geo.Point(-25.363882,131.044922)
});
```

The new API's `Map` constructor requires a base layer, here provided as `defaultLayers.normalmap`. The default base layers were pre-initialized (static) properties in the old API, but pre-initialization is no longer possible in version 3.x, because:

- the map rendering core and HERE Service Platform access are decoupled
- the HERE Map Tile API has a much larger selection of map types and variants than the old Maps API for JavaScript

To provide convenient access to the basic map types, the `Platform` object allows you to retrieve the default layers with one method call and you can then quickly select the map styles to set as base layers on the map. For more details please refer to the developer's guide of the new HERE Maps API for JavaScript (v3).

Credentials

The examples under *Initializing the Map* on page 12 reveal an important change in setting up credentials brought in by version 3.x of the Maps API for JavaScript.

Setting Credentials in 2.x

The old version of the API used the global class named `Settings` and an application provided the authentication and authorization credentials by calling `set()` on it:

```
nokia.Settings.set("app_id", "<!-- your app id -->");
nokia.Settings.set("app_code", "<!-- your token -->");
```

Setting Credentials in 3.x

In Maps API for JavaScript 3.x, the application must set the credentials on every extension that it requires. As a rule, set the credentials when initializing the `H.service.Platform` object from `service` extension, which enables to access to HERE services (Map Tiles, Search, Geocoder,

Routing, Places). However, the StreetLevel (pano) extension requires the credentials to be set on the `RenderEngine` class as well. The following code sets the credentials on both the `service.Platform` object and `map.render.panorama.RenderEngine`:

```
<!DOCTYPE html>
    <head>
        <script src="http://js.api.here.com/v3/3.0/mapsjs-core.js"></script>
        <script src="http://js.api.here.com/v3/3.0/mapsjs-pano.js"></script>
        <script src="http://js.api.here.com/v3/3.0/mapsjs-service.js"></script>
        <script>
            // Initialize platform object with credentials:
            var platform = new H.service.Platform({
                app_id: '<!-- your app id -->',
                app_code: '<!-- your token -->'
            });

            // Set credentials for street level:
            mapsjs.map.render.panorama.RenderEngine.configure('<!-- your app id -->', '<!--
 your token -->', false);
        </script>
    </head>
    ...
```

Markers

The Maps API for JavaScript 2.x offered two `Marker` classes to show point objects on the map: the `StandardMarker`, which was the default platform marker with a default icon, and `Marker` which could be customized by the user.

The new API offers the `Marker` class, which behaves very much like `Marker` in version 2.x. The main difference lies in the underlying rendering mechanism. While version 2.x renders each `Marker` instance as a DOM node on top of the map canvas, the new API renders a `Marker` onto the map canvas. In most circumstances, the two approaches produce indistinguishable results, but rendering on the canvas can be much faster.

In addition to the `Marker` class, the new API includes a class named `DomMarker`, which allows you explicitly to create markers as DOM elements on top of the canvas. Note that this does not replicate the marker rendering mechanism from the older API. Rather, it is a way to give developers full control over icons, and to support animated icons. Please refer to the developer's guide for additional information on markers.

Creating a Marker in 2.x

Below is a simple example that uses version 2.x of the API to create a marker with a user-defined icon:

```
var marker = new nokia.maps.map.Marker([52.51, 13.4], {
    icon: 'http://someurl.com/icon.png'
});
```

```
map.objects.add(marker);
```

Creating a Marker in 3.x

The current version of the API encourages the use of `H.map.Icon` and `H.map.DomIcon` objects to create a visual representation of the marker. This allows applications to re-use the same icon object with different markers:

```
var markerIcon = new H.map.Icon('http://someurl.com/icon.png');
var marker = new H.map.Marker({ lat: 52.51, lng: 13.4}, {
    icon: markerIcon
});
map.addObject(marker);
```

Icons and GFX

To support custom vector graphics as marker icons on modern as well as older browsers (such as Internet Explorer 7), version 2.x of the Maps API offered the `gfx` namespace. It allowed for rendering vector graphics independently of the underlying browser and supported tiny SVG.

The following code creates a marker with a custom icon based on SVG mark-up, using the old JavaScript API. Note that the icon constructor (`gfx.GraphicsImage`) requires parsed SVG input.

```
var svgMarkup =
      '<svg width="33" height="33" xmlns="http://www.w3.org/2000/svg">' +
      '<circle stroke="black" fill="white" cx="16" cy="16" r="16" />' +
      '<text x="16" y="20" font-size="10pt" font-family="arial" ' +
            'font-weight="bold" text-anchor="middle" fill="black" ' +
            'textContent="Hi!">Hi!</text>' +
      '</svg>';
var svgParser = new nokia.maps.gfx.SvgParser();
var markerIcon = new nokia.maps.gfx.GraphicsImage(svgParser.parseSvg(svg));

var marker = new nokia.maps.map.Marker([52.51, 13.4], {
    icon: markerIcon
});
map.objects.add(marker);
```

All modern browsers support the SVG standard, making the old API's graphics normalization system obsolete. The new API does not provide any mechanism to transfer SVG into another rendering back end (such as VML or Canvas). Instead, it allows you to use SVG directly as content for an `Icon` or `DomIcon` object.

The following code snippet uses the new API to create an icon directly from SVG and constructs a marker:

```
var svgMarkup =
      '<svg width="33" height="33" xmlns="http://www.w3.org/2000/svg">' +
      '<circle stroke="black" fill="white" cx="16" cy="16" r="16" />' +
      '<text x="16" y="20" font-size="10pt" font-family="arial" ' +
            'font-weight="bold" text-anchor="middle" fill="black" ' +
            'textContent="Hi!">Hi!</text>' +
      '</svg>';
```

```
var markerIcon = new H.map.Icon(svgMarkup);
var marker = new H.map.Marker({ lat: 52.51, lng: 13.4}, {
    icon: markerIcon
});
map.addObject(marker);
```

Draggable Markers

In version 2.x of the API, it was possible to specify `draggable` as a parameter at marker initialization. Dragging and visual feedback were implemented as part of the `Marker` object. The following code demonstrates the creation of such a marker:

```
var marker = new nokia.maps.map.StandardMarker([52.51, 13.4], {
    draggable: true  // Make the marker draggable
});
```

In the new version of the API, there is no default 'dragging' implementation for map objects in and markers. This means that, you, the user of the Maps API, must decide whether to change the position of a marker in response to drag events and implement the appropriate visual feedback.

To implement marker dragging, load the `mapsjs-mapevents` extension and set it up to enable events on map objects, then set the 'draggable' property on the marker to make sure that the events `dragstart`, `drag` and `dragend` are dispatched. Finally, define the event handlers that implement the observable effects of dragging the marker. The code below creates a draggable marker and the skeleton event handlers:

```
//setting up map and map object events
var mapEvents = new mapsjs.mapevents.MapEvents(map);

//creating marker and icon
var markerIcon = new H.map.Icon('http://someurl.com/icon.png');
var marker = new H.map.Marker({ lat: 52.51, lng: 13.4}, {
    icon: markerIcon
});

//marking marker as draggable
marker.draggable = true;
map.addObject(marker);

marker.addEventListener('dragstart', function() {
    //handle drag start here
});

marker.addEventListener('drag', function() {
    //handle drag here
});

marker.addEventListener('dragend', function() {
    //handle drag end here
});
```

To learn more about `Marker` and `DomMarker` in Maps API for JavaScript 3.x, please refer to the developer's guide.

HERE Services

The following articles explain the differences between accessing and using HERE services in versions 2.x and 3.x of the Maps API for JavaScript.

Working with HERE Services

Maps API for JavaScript version 2.x provided access to back-end services such as Search or Routing through *manager* objects, while version 3.x uses *service* objects. We explain the differences below.

Managers in 2.x

In the Maps API for JavaScript 2.x, accessing the HERE services was made possible by *manager* objects. For example, `nokia.maps.routing.Manager` was responsible for providing access to a back-end service and processing the response. The following code uses a `search.Manager`:

```
nokia.places.search.manager.geoCode({
    searchTerm: 'Invalidenstrasse 117 Berlin',
    onComplete: function(responseData, requestStatus) {
        //handling response
    }
});
```

The type of manager and the functionality it supported depended on the REST service and the Maps API edition. For example, the `nokia.maps.routing.Manager` in the Standard Edition accessed the HERE Routing API, but the same class in the Enterprise Edition retrieved routes from the HERE Enterprise Routing API.

Some of the manager objects in version 2.x could alter the response data and provide additional data structures, depending on the functionality used. Managers required callbacks to handle responses, and, as a further variation, sometimes a callback was an observer for changes to the `state` property as shown in the example below:

```
router = new nokia.maps.routing.Manager();
router.addObserver("state", function(router, key, value) {
    if (value === 'finished') {
        //handling response
    }
});
route.calculateRoute(waypointsList, modes);
```

Services in 3.x

The new Maps API introduces a consistent, flexible and easy to understand way to access HERE REST APIs via `service` objects. A service object maps directly to a HERE REST APIs and expose its features.

You can retrieve a service object from a `H.service.Platform` instance as demonstrated by the following code:

```
<script src="http://js.api.here.com/v3/3.0/mapsjs-service.js"></script>
<script>
    //initializing platform object with credentials
    var platform = new H.service.Platform({
        app_id: '<!-- your app id -->',
        app_code: '<!-- your token -->'
    });
    var placesService = platform.getPlacesService();
    var routingService = platform.getRoutingService();
</script>
```

Service objects work as direct connectors to their REST API counterparts. They provide methods to perform REST API requests and the parameters to those methods map *one-to-one* to the REST API request parameters. This direct access to the platform services allows for more complex and sophisticated queries than was possible with version 2.x of the API.

The following code uses a `placeService` object to submit a search request and a `routingService` object to request a route calculation:

```
placesService.search({
    'at': '52.5304417,13.4111201',
    'q': 'hotel',
}, function(serviceResponse) {
    //handle response
}, function() {
    //handle connection error
});

routingService.calculateRoute({
    'waypoint0': 'geo!52.5,13.4',
    'waypoint1': 'geo!52.5,13.45',
    'mode': 'fastest;car;traffic:disabled'
}, function(serviceResponse) {
    //handle response
}, function() {
    //handle connection error
});
```

Note also that in this example, both the HERE Routing API and the HERE Enterprise Routing API are provided side by side to allow you to use the full feature set from both services, including, for example, public transit routing (exclusive to the HERE Routing API) and isoline routing (exclusive to the HERE Enterprise Routing API) in the same application.

Within the new Maps API for JavaScript, the response data structures are not augmented or modified in any way. The results are passed to the response handlers as they are retrieved from the service.

Adding a Route

The Maps API for JavaScript 3.x does not modify data returned from the REST APIs nor does it create data structures on top of the service response. Therefore, in some cases, there are extra steps involved when an application must display the data. This is necessary, for example, when a route retrieved from the HERE Routing API is to be shown on the map.

Adding a Route in 2.x

In version 2.x of the API, the `nokia.maps.routing.component.RouteResultSet` object was used to create a map representation of the route. This object was responsible for creating a `container` which held a polyline that could be added to the map to display the route. In addition, an instance of `nokia.maps.routing.WaypointParameterList` had to be created to provide a list of waypoints to the routing manager when requesting a route. This introduced additional data structures which were strictly related to the routing service:

```
var router = new nokia.maps.routing.Manager(); // Create a route manager.
// Set a handler to dispaly the route:
router.addObserver("state", function (observedRouter, key, value) {
    if (value == "finished") {
        var routes = observedRouter.getRoutes();

        // Create the default map representation of the route:
        var mapRoute = new nokia.maps.routing.component.RouteResultSet(routes[0]).container;
        map.objects.add(mapRoute);

        // Zoom to the bounding box of the route:
        map.zoomTo(mapRoute.getBoundingBox(), false, "default");
    } else if (value == "failed") {
        alert("The routing request failed.");
    }
});

// Create waypoints:
var waypoints = new nokia.maps.routing.WaypointParameterList();
waypoints.addCoordinate(new nokia.maps.geo.Coordinate(52.516222633265954,
 13.3889009369434));
waypoints.addCoordinate(new nokia.maps.geo.Coordinate(52.51717584105763,
 13.395129026281722));

var modes = [{
    type: "shortest",
    transportModes: ["car"],
    options: "avoidTollroad",
    trafficMode: "default"
}];
router.calculateRoute(waypoints, modes);
```

Adding a Route in 3.x

The new version of the API introduces no additional data structures. Instead, the API user needs to set the request parameters for the routing service method `calculateRoute()` as specified in the

Routing API documentation, and then create a `Polyline` object from the response returned by the service. This is shown in the example below:

```
var routingService = platform.getRoutingService();
routingService.calculateRoute({
    waypoint0: 'geo!52.516222633265954,13.3889009369434',
    waypoint1: 'geo!52.51717584105763, 13.395129026281722',
    mode: "fastest;car;traffic:disabled",
    representation: 'display'
  }, function(result) {
    // Once a route object is returned, the application turns the shape of
    // the route into a polyline and adds a marker for the start and end points:
    var strip = new H.geo.Strip(),
        shape = result.response.route[0].shape,
        coords,
        i,
        l = shape.length;

    for(i = 0; i < l; i++) {
      coords = shape[i].split(',');
      strip.pushLatLngAlt(+coords[0], +coords[1]);
    }
    var polyline = new H.map.Polyline(strip, {
      style: {
        lineWidth: 5
      }
    });

    // Create an icon element for the marker:
    var startPosition = result.response.route[0].waypoint[0].mappedPosition;
    var startMarker = new H.map.Marker({
          lat: startPosition.latitude,
          lng: startPosition.longitude
        });

    var endPosition = result.response.route[0].waypoint[1].mappedPosition;
    var endMarker = new H.map.Marker({
          lat: endPosition.latitude,
          lng: endPosition.longitude
        });

    map.addObject(startMarker);
    map.addObject(endMarker);
    map.addObject(polyline);
 }, function() {
  // Code to handle a connection error here.
 });
```

Extended Functionality

Maps API for JavaScript version 2.x used a *component* model to provide map-centric functionality extensions such as behavior or info bubbles. The HERE Maps API for JavaScript version 3.x no longer makes use of the component model and instead relies on extension libraries to provide all map functionality beyond pure rendering.

Components in 2.x

Components providing extended functionality were loaded by default as part of the API and were available immediately after loading `http://js.api.here.com/se/2.5.4/jsl.js`.

The following example adds *behavior* and *ZoomBar* to the map, using version 2.x. The code initializes the components `nokia.maps.map.components.Behavior` and `nokia.maps.map.components.ZoomBar` and then passes them to the map object at initialization:

```
...
<script src="http://js.api.here.com/se/2.5.4/jsl.js" type="text/javascript"
 charset="utf-8"></script>
...
<script>
    var map = new nokia.maps.map.Display(
        document.getElementById("mapContainer"), {

            // List of components to be added to the map:
            components: [
                new nokia.maps.map.component.Behavior(),
                new nokia.maps.map.component.ZoomBar()
            zoomLevel: 10,
            center: [52.51, 13.4]
        }
    );
</script>
```

Extensions in 3.x

If a 3.x application requires functionality additional to that provided by the `core` module, it must load the appropriate extensions explicitly and individually – only those extensions whose functionality the application needs.

An extension provides its own namespace and adds its functionality *on top* of the `core`.

To add `Behavior` and `ZoomBar` as in the 2.x example above, load the extensions `mapsjs-mapevents`, which provides the events system and map behavior, and `mapsjs-ui` which provides the default user interface. This is shown by the code below:

```
...
<!-- Include mapsjs core functionality -->
<script src="http://js.api.here.com/v3/3.0/mapsjs-core.js" type="text/javascript"
 charset="utf-8"></script>

<!-- Include mapsjs service extension -->
<script src="http://js.api.here.com/v3/3.0/mapsjs-core.js" type="text/javascript"
 charset="utf-8"></script>

<!-- Include mapsjs UI and mapevents extensions -->
<script src="http://js.api.here.com/v3/3.0/mapsjs-mapevents.js" type="text/javascript"
 charset="utf-8"></script>
<script src="http://js.api.here.com/v3/3.0/mapsjs-ui.js" type="text/javascript"
 charset="utf-8"></script>
...
<script>
    var map = new H.Map(document.getElementById('map-container'), defaultLayers.normal.map,
 {
```

```
      zoom: 4,
      center: new H.geo.Point(52.51, 13.4)
    });

    // Add the map events functionality:
    var mapEvents = new mapsjs.mapevents.MapEvents(map);

    // Add behavior (on top of map events):
    var behavior = new mapsjs.mapevents.Behavior(mapEvents);

    // Add the UI with just the zoom bar
    var ui = new mapsjs.ui.UI(map, {
      zoom: true
    });

</script>
```

Compared to 2.x, version 3.x does, in fact, require additional steps to set up extensions. However, this offers greater flexibility to you as a developer, because all extensions are purely optional and – as their name suggests – they *extend* (that is *build on*) the functionality of `mapsjs-core` (and/or of any other extensions on which they depend). A further benefit of this architecture is that it allows you to create your own extensions and to replace existing ones to meet specific requirements.

The following table summarizes the available extension modules and the functionality they provide.

Table 3: Extensions in Maps API version 3.x

Extension File	Functionality
mapsjs-service.js	Provides access to HERE platform (map tiles, REST APIs)
mapsjs-mapevents.js	Implements map and map object events and behavior
mapsjs-ui.js	Implements the default UI controls
mapsjs-pano.js	Provides StreetLevel view functionality
mapsjs-clustering.js	Implements clustering functionality
mapsjs-data.js	Implements KML and GeoJSON import functionality

Event Handling

Maps API for Javascript version 3.x introduces a new event system. It is more streamlined than its 2.x counterpart and an application using must load an extension.

Event Handling in 2.x

Version 2.x of the Maps API provided its own standardized event framework which could be used for map and map objects in addition to any DOM elements available on the page. Every `EventTarget` implemented the method `addListener()` and `removeListener()` to manage event handlers. This event system supported mouse events and touch events (MS pointer events were mapped to

the corresponding touch events). The code below illustrates this, adding event listeners to the map, a map marker and the map container.

```
// Add an event handler to the map:
map.addListener('click', function(e) {
   // Handle click
});

var marker = new jsl.map.Marker(map.center);
marker.addListener('mousedown', function(e) {
    // handle mousedown
});

var eventEl = new nokia.maps.dom.EventTarget(document.getElementById('mapContainer'));
eventEl.addListener('touchmove', function(e) {
    // handle touchmove
});
```

Event Handling in 3.x

In version 3.x of the API, the event system is provided via an extension. It must be explicitly loaded (if it is required) and set up for the map object. The new event system works only for the map instance and any map objects added to it, but cannot be applied to DOM elements.

Due to the extended support for mobile devices, the new event system does not provide as many different events as the system in version 2.x, but it normalizes events (mouse, touch, pointer) from different platforms in its own `pointer` *events setup*. It treats all native events as pointer events, but you can distinguish between mouse and touch by examining the pointer's `type` property. The following code uses version 3.x of the API to enable events and add event handlers to the map and a marker:

```
// Set up events for the map:
var mapEvents = new H.mapevents.MapEvents(map);

// Add a tap listener and callback:
map.addEventListener('tap', function(e) {
    if (e.currentPointer.type === 'mouse') {
        // handle a click
    } else if (e.currentPointer.type = 'touch') {
        // handle a tap
    }
});

// Add a pointer down listener and callback:
marker = new H.map.Marker(map.getCenter());
marker.addEventListener('pointerdown', function(e) {
    if (e.currentPointer.type === 'mouse') {
        // handle mousedown
    } else if(e.currentPointer.type === 'touch') {
        // handle touchdown
    }
});
```

User Interface

Version 3.x of the Maps API for JavaScript uses an extension to provide user interface elements, abandoning the component model of version 2.x. This affects also info bubbles, which belong to the UI extension in version 3.x.

UI elements

In Maps API for JavaScript 2.x every UI element was a separate `component`. All the UI components required by an application needed to be initialized and then passed to the map constructor along with other components. This is shown in the example below:

```
var map = new nokia.maps.map.Display(
    document.getElementById("mapContainer"), {
        //list of components which will be added to the map
        components: [
            new jsl.map.component.ZoomBar(),
            new jsl.map.component.ScaleBar(),
            new jslocation.map.component.TypeSelector()
        zoomLevel: 10,
        center: [52.51, 13.4]
    }
);
```

In Maps API for JavaScript 3.x, all UI functionality is present only when the application loads the `mapsjs-ui` extension. UI elements are no longer available as `components`, but they can be specified when initializing the UI objects. The following code loads the user interface, using version 3.x of the API:

```
var mapTypes = platform.createDefaultLayers();
var ui = new mapsjs.ui.UI(map, {
  zoom: true,
  scalebar: true,
  mapsettings: {
    entries: [
        {'name': 'Normal', 'mapType': mapTypes['normal']},
        {'name': 'Satellite', 'mapType': mapTypes['satellite']},
        {'name': 'Terrain', 'mapType': mapTypes['terrain']}
    ]
  }
});
```

Info Bubbles

In Maps API for JavaScript 2.x, a separate component provided `infobubbles`. The component was added to the map and worked as a factory for placing the actual info bubbles on the map display. This was consistent with the handling of all other UI elements.

The following code demonstrates how an info bubble was created and added to the map in version 2.x:

```
var infoBubbles = new nokia.maps.map.component.InfoBubbles();
var map = new nokia.maps.map.Display(mapContainer, {
    center: [52.51, 13.4],
    zoomLevel: 10,
    components: [
        //adding infobubbles component to the map
        infoBubbles,
        new nokia.maps.map.component.Behavior(),
        new nokia.maps.map.component.ZoomBar()
    ]
});
var bubble = infoBubbles.openBubble('<b>the content of the bubble</b>', map.center);
```

In Maps API for JavaScript 3.x, info bubbles are part of the `mapsjs-ui` extension and can be only added to the map if the UI is initialized and set for the current map object:

```
// Add UI to the map object:
var ui = new mapsjs.ui.UI(map);

// Initialize infobubble
var bubble = new H.ui.InfoBubble(map.getCenter(), {
        content: '<b>Hello World!</b>'
    });

// Add info bubble to the UI:
ui.addBubble(bubble);
```